Aspect-Oriented User Requirements Notation: Aspects in Goal and Scenario Models

Gunter Mussbacher

SITE, University of Ottawa, 800 King Edward, Ottawa, ON, K1N 6N5, Canada
gunterm@site.uottawa.ca

Abstract. Technologies based on aspects and applied at the early stages of software development allow requirements engineers to better encapsulate crosscutting concerns in requirements models. The Aspect-oriented User Requirements Notation (AoURN) extends the User Requirements Notation (URN) with aspects and thus unifies goal-oriented, scenario-based, and aspect-oriented concepts in one framework. Minimal changes to URN ensure that requirements engineers can continue working with goal and scenario models expressed in a familiar notation. At the same time, concerns in goal and scenario models, regardless of whether these concerns crosscut or not, can be managed across model types. Typical concerns in URN are non-functional requirements (NFRs), use cases, and stakeholder goals. As AoURN expresses concern composition rules with URN itself, it is possible to describe rules in a highly flexible way that is not restricted by any specific composition language. Aspects can improve the modularity, reusability, scalability, and maintainability of URN models. Considering the strong overlap between NFRs and crosscutting concerns, aspects can help bridge the gap between goals and scenarios. On the other hand, Early Aspects (EA) research can benefit from a standardized way of modeling concerns with AoURN.
Keywords: Aspect-oriented Requirements Engineering, Aspects, Use Case Maps, Goal-oriented Requirement Language, User Requirements Notation.

1 Introduction

By the end of the 1990s, Aspect-Oriented Programming (AOP) [10] allowed software engineers to better encapsulate, at the implementation level, crosscutting concerns (i.e. aspects) which are notoriously difficult to modularize with a single dominant modularization technique alone (e.g. with object-oriented concepts). During the last decade, the research community has shifted its emphasis more to Early Aspects (EA) [8] by investigating ways of addressing crosscutting concerns in requirements and design models. Two of the most common requirements engineering models are goal-oriented and scenario-based models. The User Requirements Notation (URN) [2, 16, 17] is the first and currently only standardization effort that combines goal and scenario models in one language. The Aspect-oriented URN (AoURN) [11–14] aims to extend URN with aspect

concepts to better manage crosscutting concerns in goal and scenario models. AoURN unifies goal-oriented, scenario-based, and aspect-oriented concepts in one framework.

AoURN consists of Aspect-oriented Use Case Maps (AoUCM), first introduced in [11], and the Aspect-oriented and Goal-oriented Requirement Language (AoGRL), first introduced in [13]. A detailed description of AoUCM and its matching and composition algorithms is available in [12], while AoURN's flexible and exhaustive composition rules based on URN itself are discussed in [13, 14]. Qualitative and quantitative assessments of the modularity, reusability, scalability, and maintainability of AoURN are available in [12, 14] and [13], respectively. This paper presents AoURN as a whole for the first time, further aligning AoUCM and AoGRL with each other, and discusses the relationship of aspects in goal and scenario models.

In the remainder of this paper, Sect. 2 gives an overview of URN and related work on EA. Section 3 describes AoURN, while Sect. 4 discusses the relationship of aspects in goal and scenario models as illustrated by a sample AoURN model. Section 5 concludes the paper and identifies future work.

2 Background

2.1 User Requirements Notation

The User Requirements Notation (URN) [2, 16, 17], a standardization effort of the International Telecommunication Union (ITU-T Z.150 Series), contains two complementary modeling languages for goals and scenarios. The Goal-oriented Requirement Language (GRL) is a visual modeling notation for business goals and non-functional requirements (NFRs) of many stakeholders, for alternatives to be considered, for decisions that were made, and for rationales that helped make these decisions. GRL supports reasoning about goals and NFRs with the help of GRL *strategies*. A strategy describes a particular configuration of alternatives in the GRL model. An *evaluation mechanism* propagates these low-level decisions regarding alternatives to satisfaction ratings of high-level stakeholder goals and NFRs. A reusable goal model is called a *GRL catalogue*.

Use Case Maps (UCMs) are a visual scenario notation that focuses on the causal flow of behavior optionally superimposed on a structure of components. UCMs depict the interaction of architectural entities while abstracting from message and data details. UCMs support the definition of *scenarios* including pre- and post-conditions. A scenario describes a specific path through the UCM model where only one alternative at any choice point is taken. Given a scenario definition, a *traversal mechanism* can highlight the scenario path or transform the scenario into a message sequence chart (MSC). Essentially, the traversal mechanism turns the scenario definitions into a test suite for the UCM model.

URN links indicated by small triangles can link any two URN model elements. In particular, links from GRL models to UCM models establish traceability between goal and scenario models in URN. URN is the first and currently only

standardization effort to address explicitly, in a graphical way, and in one language goals and scenarios, and the links between them. Current tool support for URN is available with the Eclipse-based jUCMNav tool [15]. Over the last decade, GRL and UCMs have successfully been used for service-oriented, concurrent, distributed, and reactive systems such as telecommunications systems, agent systems, e-commerce systems, operating systems, health information systems, and business process modeling [16].

2.2 Goal-Oriented Requirement Language

The *Goal-oriented Requirement Language* (GRL) [2, 15, 16] combines the Non-Functional Requirements (NFR) framework [6] and i* framework [19] to support reasoning about goal models. The syntax of GRL (Fig. 1) is based on the syntax of the i* framework. A GRL goal graph is an AND/OR graph of *intentional elements* that optionally reside within an *actor boundary*. An *actor* represents a stakeholder of the system. A goal graph shows the high-level business goals and non-functional requirements of interest to a stakeholder and the alternatives for achieving these high-level elements. A goal graph also documents rationales (*beliefs*) important to the stakeholder.

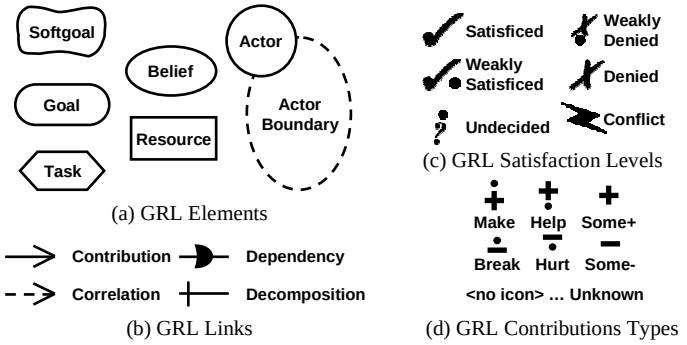

Fig. 1. Basic Elements of GRL Notation

Various kinds of *links* connect the elements in a goal graph, allowing an element to be decomposed into sub-elements, indicating desired impacts and side effects of one element on another element, and modeling relationships between actors (one actor depending on another actor for something). A more complete coverage of the notation elements is available in [2, 15].

From the NFR framework, GRL borrows the notion of an evaluation mechanism that supports reasoning about the goal graph. The decisions of stakeholders are typically documented in the goal graph by the assignment of satisfaction levels (Fig. 1.c) to alternatives (e.g. the chosen alternative is set to Satisficed whereas all other alternatives are set to Denied). Based on these initial settings and the various links with various contribution types (Fig. 1.d), the satisfaction ratings are propagated to higher-level goals and non-functional requirements of stakeholders. jUCMNav keeps track of these initial settings separate from goal graphs in *strategies*. Several strategies can be defined for a goal model, allowing

trade-off analyses to be performed by exploring and comparing various configurations of alternatives. GRL also takes into account that not all high-level goals and non-functional requirements are equally important to the stakeholder. Therefore, jUCMNav supports the definition of *evaluation attributes* for intentional elements such as *priority* (high, medium, low, none), which are also taken into account when evaluating strategies for the goal model [15].

2.3 Use Case Maps

Use Case Maps (UCMs) [2, 4, 16] are ideally suited for the description of functional requirements and, if desired, high-level design. Paths describe the causal flow of behavior of a system (e.g. one or many use cases). By superimposing paths over components, the architectural structure of a system can be modeled. In general, components describe any kind of structural entity at any abstraction level (e.g. classes or packages but also systems, actors, sub-systems, objects, aspects, hardware). As many scenarios and use cases are integrated into one combined UCM model of a system, it is possible to use UCM specifications as a base for further analysis. Undesired interactions between scenarios can be detected, performance implications can be analyzed, testing efforts can be driven based on the UCM model, and various architectural alternatives can be analyzed. For further information, the reader is referred to the URN Virtual Library [16].

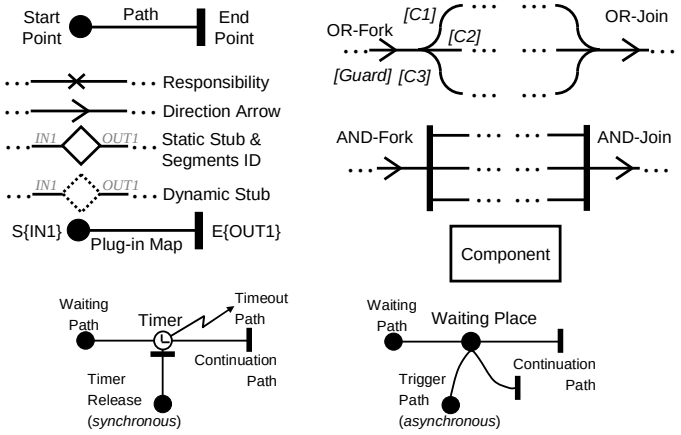

Fig. 2. Basic Elements of UCM Notation

The basic elements of the UCM notation are shown in Fig. 2. A *map* contains any number of *paths* and *components* (structural elements). UCM models can be decomposed using *stubs* which contain sub-maps called *plug-ins*. Plug-in maps are reusable units of behavior and structure. *Plug-in bindings* define the continuation of a path on a plug-in by connecting in-paths and out-paths of a stub with start and end points of its plug-ins, respectively. A stub may be *static* which means that it can have at most one plug-in, whereas a *dynamic* stub may have many plug-ins which may be selected at runtime. A *selection policy* decides which plug-ins of a dynamic

stub to choose at runtime. A more complete coverage of the notation elements is available in [2, 4].

2.4 Overview of Related Work on Early Aspects

As the Aspect-oriented User Requirements Notation (AoURN) [11–14] makes use of goal and scenario models, we will briefly review aspect-oriented approaches to requirements engineering that apply to goal and scenario models. For a comparison of these approaches to AoURN or an introduction to aspect concepts, see [11–14].

In *Aspect-Oriented Software Development (AOSD) with Use Cases* [5], Jacobson and Ng consider a well-written use case a concern. Extension points identify a step in a use case where an extension may occur. Pointcuts in other use cases reference such extension points. Aspects allow use cases to be encapsulated throughout the software development lifecycle.

In *Scenario Modeling with Aspects* [5], Whittle and Araújo model aspectual scenarios with sequence-diagram-like *interaction pattern specifications* (IPS) and *state machine pattern specifications* (SMPS). IPS and SMPS define roles which can be bound to elements in other sequence diagrams and state machines.

In the *Aspectual Use Case Driven Approach* [5], Araújo and Moreira visualize how crosscutting non-functional requirements captured with templates are linked to functional requirements (use case diagrams or sequence diagrams). *Activity pattern specifications* (APS) similar to the aforementioned IPS and SMPS are used. In addition, new use-case relationships allow the impact of one use case on another to be described (restricting or contributing positively/negatively).

Barros and Gomes [3] apply aspect-orientation to activity diagrams (AD) by describing ways to merge stereotyped nodes in one AD with nodes in another. Whittle *et al.* [18] propose a metamodel-based aspect composition technique that uses graph transformation formalisms. This approach can be applied to any model for which a metamodel has been defined. In the UCM community, de Bruin and van Vliet [7] allow behaviour to be added before and after a UCM by explicitly adding "Pre" and "Post" stubs to the UCM. Yu *et al.* [20] identify aspects in goal models based on relationships between functional and non-functional goals. Goal aspects are proposed to address scalability issues but it is pointed out that the goal aspects' syntax still requires further research. Alencar *et al.* [1] identify aspects in i* models. Their extensions to aspect-oriented concepts, however, do not fully separate concerns from other concerns. Kaiya and Saeki [9] propose a pattern-based technique to compose viewpoints. The approach lacks formalization and limits its composition to a simple combinatorial approach instead of more powerful pointcut expressions.

3 Aspect-Oriented User Requirements Notation

The Aspect-oriented User Requirements Notation (AoURN) [11–14] extends the User Requirements Notation (URN) by defining a *joinpoint model* for the Goal-oriented Requirements Language (GRL) and Use Case Maps (UCMs). All nodes

of GRL graphs or UCMs optionally residing within the boundary of an actor or component are deemed to be joinpoints (except for purely visual elements such as direction arrows). Joinpoints can be matched by pointcut expressions. Therefore, pointcut expressions for AoURN models can identify any URN node which in turn can be transformed by the aspect. Joinpoints matched by a pointcut expression are indicated with small, filled diamonds called *aspect markers*, identifying the insertion points for aspectual behavior in the base model.

Pointcut expressions are defined on *pointcut diagrams (graphs and maps)* that are matched against the rest of the model. Pointcut diagrams are standard URN diagrams, allowing the requirements engineer to continue working with familiar models. Pointcut diagrams can also be parameterized for increased matching power by allowing names of modeling elements to contain wildcards ("*") and logical expressions (containing "and", "or", and "not"). The goals, behavior, and structure of aspects are defined on *advice diagrams (graphs and maps)* which are loosely coupled to pointcut diagrams, allowing advice diagrams and pointcut diagrams to be reused independently from each other. Again, advice diagrams are standard URN diagrams. Flexible composition rules are defined with URN itself and are therefore as expressive as URN and not restricted by the capabilities of any particular pointcut language (which for example could only allow standard before/after/around rules).

A *concern* is simply an organizational construct that contains all URN diagrams required to describe a concern. In the case of an *aspect* (i.e., a crosscutting concern), it contains (a) any number of advice diagrams and (b) any number of pointcut diagrams required to describe the aspect. Note that the order aspects are applied to an AoURN model can be specified (not discussed here due to space constraints; see [13]).

Note that in Fig. 3, Fig. 4, and Fig. 5 the aspect markers on the pointcut map, the long-dash-dot-dotted lines without arrowheads, and the dashed arrows are not part of the AoURN notation but have been added to the figures to clearly indicate the connection between the pointcut expression and the base model, the mapping of the pointcut expression to the base model, and the plug-in bindings for the UCM model, respectively. Any AoURN tool must retain the mappings and aspect markers in order to navigate and reason about the AoURN model in an aspect-oriented way. For example, double-clicking on an aspect marker presents a list of all matched pointcut expression to the requirements engineer. Selecting one of them then takes the requirements engineer to the advice diagram where the relevant portion of the diagram is highlighted.

3.1 Aspect-oriented and Goal-oriented Requirement Language

Aspect-oriented GRL (AoGRL) [13] adds support for aspect-oriented modeling to GRL. Advice graphs are very similar to the notion of GRL catalogues if the catalogue describes the goal model of only one concern. AoGRL adds the ability to easily include GRL catalogues multiple times into a GRL model by visually specifying a pointcut expression on a pointcut graph. All nodes and links in a pointcut expression are identified by *pointcut markers* (Fig. 3). Actors are

implicitly included in the pointcut expression when an element of a pointcut expression resides within the boundary of the actor. For example, the pointcut graph in Fig. 3 matches against all goal graphs that contain a softgoal Stakeholder Goal C1 which is an OR decomposition of a softgoal ending with Goal B and has a correlation with another softgoal, all of which have to reside within an actor.

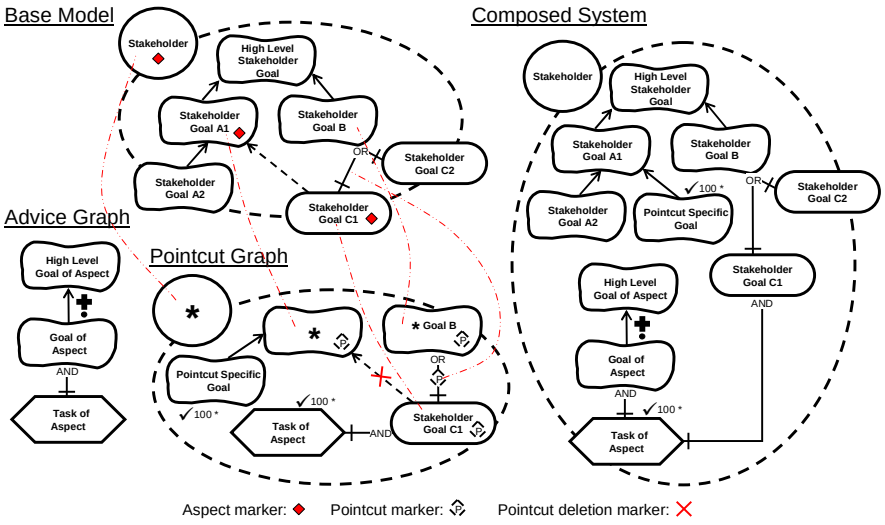

Fig. 3. Basic Elements of AoGRL Notation

Pointcut graphs contain not just the pointcut expression, but also other elements not identified with a pointcut marker. These elements either reference elements in the aspect's advice graph or are new pointcut-specific elements introduced by the aspect. Connecting these elements with elements of the pointcut expression defines the *composition rule* for the aspect. The composition rule is applied to each joinpoint matched by the pointcut expression. The composition rule may also state that matched elements should be removed with the help of the *pointcut deletion marker* (Fig. 3). Note that the ability to mark elements in a pointcut expression is the only extension required for the jUCMNav tool in order to specify AoGRL models. The composition rule in Fig. 3 stipulates that Task of Aspect and Pointcut Specific Goal are to be connected to Stakeholder Goal C1 and the matched softgoal, respectively. Furthermore, the correlation between Stakeholder Goal C1 and the matched softgoal is to be removed. The composed system is shown on the right side of Fig. 3 as a traditional GRL graph.

3.2 Aspect-oriented Use Case Maps

Aspect-oriented Use Case Maps (AoUCM) [11, 12, 14] extend UCMs with the ability to specify *pointcut stubs*, thus enabling aspect-oriented modeling. Point-

cut stubs (Fig. 4) are structurally the same as dynamic stubs but have a slightly different semantic meaning (indicated by the P in the dynamic stub symbol). While dynamic stubs contain plug-in maps that further describe the structure and behavior of a system, pointcut stubs contain zero or more pointcut maps that visually describe pointcut expressions. This is the only semantic change to traditional UCMs required in order to model aspects with AoUCM. Note that the ability to mark stubs as a pointcut stub is also the only extension required for the jUCMNav tool in order to specify AoUCM models.

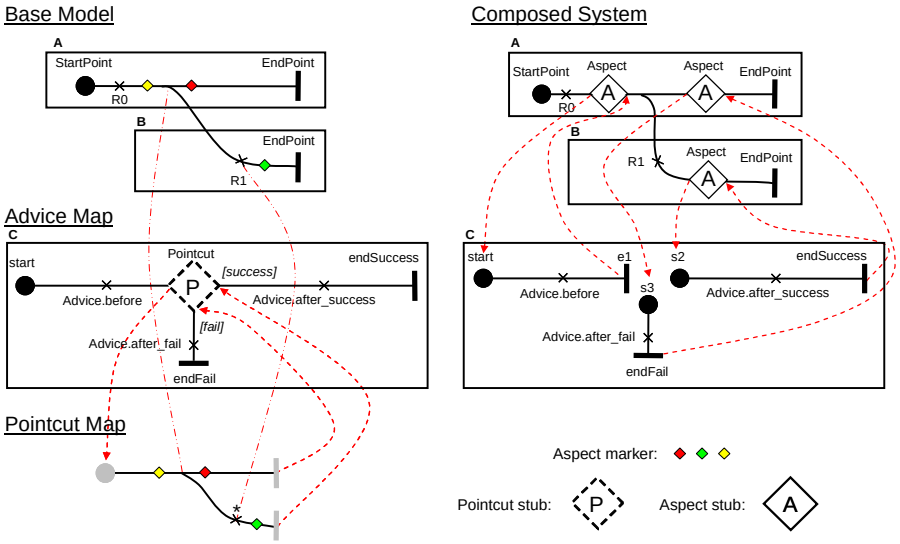

Fig. 4. Basic Elements of AoUCM Notation

An advice map (Advice Map in Fig. 4) describes the behavior and structure of an aspect and differs from a traditional map (Base Model in Fig. 4) only in that it contains one or more pointcut stubs. For example, the pointcut stub in Fig. 4 contains a pointcut map that matches against all maps that contain an OR-fork followed by a responsibility on at least one branch. Start and end points without labels on a pointcut map are not included in the match but only denote the beginning and end of the pointcut expression to be matched (therefore they are shown in gray in Fig. 4). Although not shown here, a pointcut map may contain UCM components. In fact, any behavioral or structural UCM modeling element can be used on a pointcut map, allowing a wide array of partial maps to be matched.

The way a pointcut stub is connected to the rest of the advice map visually defines the *composition rule* for the aspect. Pointcut expressions and composition rules are therefore clearly separated. Figure 4 shows that Advice.before must be inserted before the joinpoint identified by the expression in the pointcut stub.

Advice.after_returning and Advice.after_throwing are inserted after the identified joinpoint in the success case and fail case, respectively. In addition to the simple before and after composition rules, AoUCM can also easily model around, loop, concurrency, and interleaving composition rules [14]. The composed system is shown in a traditional UCM model on the right side of Fig. 4 with the help of *aspect stubs* (regular stubs that are used to insert aspect behavior). For more details on the visualization of the composed system and the techniques used to carry out composition in AoUCM see [12].

4 On the Relationship of Goal and Scenario Aspects

AoURN's ability to encapsulate NFRs as well as use cases in both model types helps bridge the gap between goal and scenario models. This gap is further narrowed by URN traceability links between modeling elements of goal and scenario aspects. Fig. 5 shows a simplified version of the YKeyK model. YKeyK stands for Your Key Knows, a system that allows drivers to find their car in a car park by following directions shown on a small display of the car key.

The AoURN model contains three concerns, one each for the Driver, YKeyK, and Car Park stakeholders, and two aspects, one for the use case UC002 and one for the Performance NFR. A stakeholder's concerns (i.e., the goals and alternatives related to a stakeholder) are modeled separately for each stakeholder on a goal graph (Fig. 5.a to c). The use case aspect contains the UC002 Pointcut Graph and the UC002 UCMs (Fig. 5.d, f, and h). The NFR aspect contains the Performance Pointcut Graph and the Performance Catalogue (Fig. 5.e and g).

The use case aspect in the goal model directly crosscuts the three stakeholder concerns while the performance aspect crosscuts the stakeholder concerns via the shown use case (as well as other use cases in the complete model). In the goal model, the use case pointcut expresssion matches the three intentional elements in Fig. 5.a to c as indicated by the aspect markers, thus adding the use case aspect to the stakeholder concerns. The performance pointcut expression matches the two dependencies in Fig. 5.d, thus adding the Handle Response Time task to the target of the dependency (the goals with the aspect markers in the YKeyK and Car Park actors of Fig. 5.d). In addition, the Performance Pointcut Graph stipulates that Performance goals have to be added to the actors in which the source and target of the dependency reside. Finally, URN links trace the Driver and YKeyK actors to their UCM components and the three tasks in Fig. 5.d to elements in their UCMs in Fig. 5.f and h.

The use case aspect in the scenario model contains three UCMs. Search for Car describes the main purpose of this aspect. The search, however, requires topology information about the car park which is transmitted when the car enters the car park. As this information is only required by the search capability but must take place during the execution of the Enter Car Park use case (not shown due to space constraints), the required responsibilities are added with the help of an aspect (Fig. 5.f) to the setup stage after the prepare price list responsibility in the Enter Car Park use case.

The use case in the YKeyK example shows that an aspect in AoGRL can be traced to an aspect in AoUCM. Often, use cases are not crosscutting in a scenario model but rather peers to each other. When modeled in AoGRL, however, use cases are usually crosscutting. With AoURN, the crosscutting use cases can be properly encapsulated even in goal models.

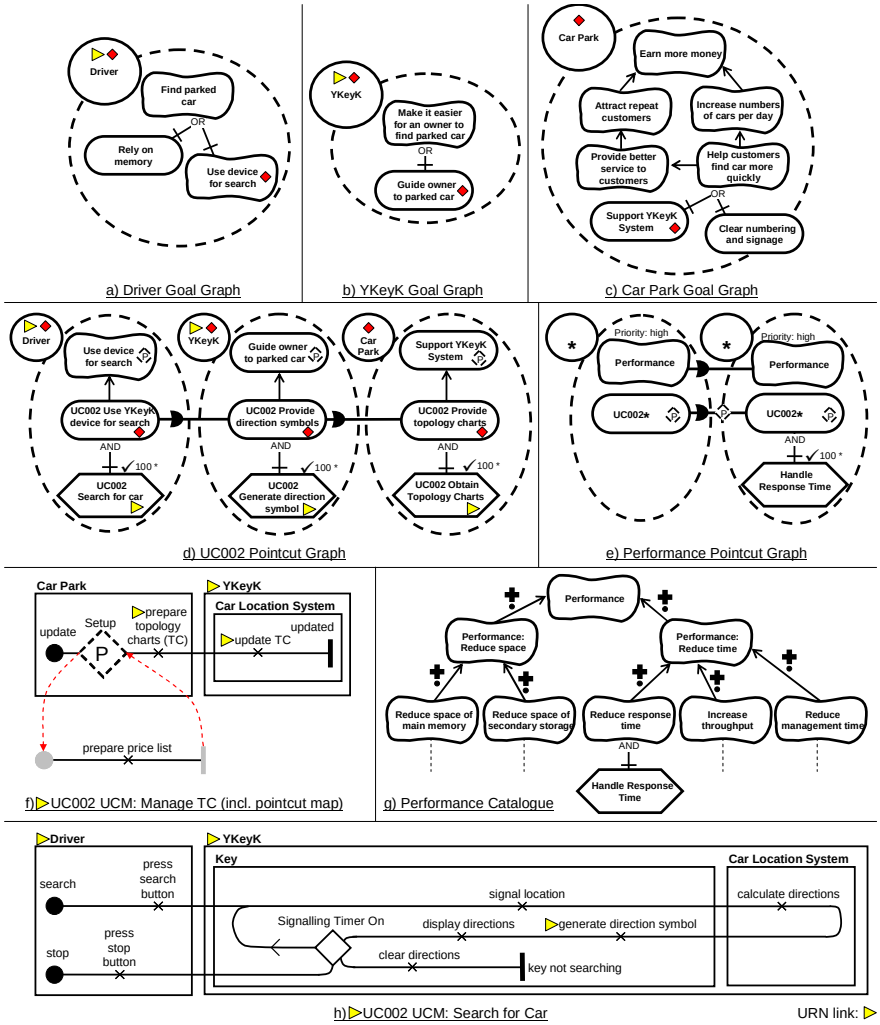

Fig. 5. AoURN Model of YKeyK System

Though not shown in the simplified YKeyK model, the performance aspect can also be traced to the scenario model. In this case, however, there are many ways of achieving performance improvements (e.g. caching, concurrency, ...).

Each should be modeled as a separate concern in AoUCM. Clearly, there is a one-to-many relationship between crosscutting concerns in AoGRL and AoUCM. Cases, however, exist where aspects in AoGRL are not traced to any aspects in AoUCM and vice versa. AoUCM can model non-functional requirements that can be expressed as scenarios. AoGRL, however, can model a much larger class of non-functional requirements. For example, the quality of a software product may be modeled in an AoGRL graph and it may be decided that inspections are the solution for this goal. This solution cannot be modeled in AoUCM (or UCM for that matter). Similarly, new aspects may appear in AoUCM models since there is a difference in the abstraction levels of goal models and scenario models. Whatever the relationship between aspects in goal and scenario models may be, AoURN aspects can encapsulate crosscutting concerns across the two model types and URN links allow keeping track of these relationships.

5 Conclusion and Future Work

The Aspect-oriented User Requirements Notation (AoURN) extends the abstract syntax, the concrete syntax, and the semantics of URN with aspect-oriented concepts, hence unifying goal, scenario, and aspect concepts in one framework. AoURN helps clarify the relationship between aspects in goal and scenario models, allowing clearer management of major concerns in goal models that crosscut scenario models and vice versa (e.g. non-functional requirements and use cases). AoURN uses flexible composition rules that are only limited by the expressiveness of URN itself (as opposed to a particular composition language). While some support for aspect-oriented modeling is available in the jUCMNav tool as a proof of concept, further support for matching and composition is currently at the prototyping stage and has yet to be officially released in jUCMNav. Such support will ensure that aspect markers are added to AoURN models and that AoURN models can be navigated with the help of aspect and pointcut markers. Furthermore, additional qualitative assessments of AoURN with respect to desirable properties of aspect-oriented requirements models and quantitative assessments of AoURN based on metrics for aspect-oriented requirements models adapted for URN and AoURN are required. Finally, the relationship of aspects, GRL strategies, and UCM scenarios should be clarified and the applicability of advanced URN research to AoURN should be investigated (e.g. feature interaction, business process modelling, performance analysis, product lines, or modeling support for inherently existing communication aspects in UCM).

Acknowledgments. This research was supported by NSERC, through its programs of Discovery Grants and Postgraduate Scholarships, and by ORNEC.

References

1. Alencar, F., Moreira, A., Araújo, J, Castro, J., Silva, C., and Mylopoulos, J.: Towards an Approach to Integrate i* with Aspects. *8th Intl. Bi-Conf. Wksh. on Agent-Oriented Inf. Systems (AOIS-2006)* at CAiSE'06, Luxembourg (June 2006)

2. Amyot, D.: Introduction to the User Requirements Notation: Learning by Example. *Computer Networks*, Vol. 42(3), pp 285-301 (21 June 2003)
3. Barros, J.-P. and Gomes, L.: Toward the Support for Crosscutting Concerns in Activity Diagrams: a Graphical Approach. *Workshop on Aspect-Oriented Modelling* (held with UML 2003), San Francisco, California, USA (October 2003)
4. Buhr, R.J.A. and Casselman, R.S.: *Use Case Maps for Object-Oriented Systems*. Prentice-Hall (1996)
5. Chitchyan, R. et al.: *Survey of Analysis and Design Approaches*. AOSD-Europe Report ULANC-9 (May 2005), http://www.aosd-europe.net/deliverables/d11.pdf (accessed November 2007)
6. Chung, L., Nixon, B.A., Yu, E., and Mylopoulos, J.: *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, Dordrecht, USA, 2000.
7. de Bruin, H. and van Vliet, H.: Quality-Driven Software Architecture Composition. *Journal of Systems and Software*, Vol. 66(3), pp 269-284 (15 June 2003)
8. *Early Aspects website*. http://www.early-aspects.net/ (accessed November 2007)
9. Kaiya, H. and Saeki, M.: Weaving Multiple Viewpoint Specifications in Goal-Oriented Requirements Analysis. *11th Asia-Pacific Software Eng. Conf. (APSEC 2004)*, Busan, Korea, IEEE Computer Society Press, pp 418-427 (Nov. 2004)
10. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., and Irwin, J.: Aspect-Oriented Programming. *ECOOP'97–Object Oriented Programming, 11th European Conference*, LNCS 1241, pp 220-242, Springer (1997)
11. Mussbacher, G., Amyot, D., and Weiss, M.: Visualizing Aspect-Oriented Requirements Scenarios with Use Case Maps. *International Workshop on Requirements Engineering Visualization (REV 2006)*, Minneapolis, USA (September 2006)
12. Mussbacher, G., Amyot, D., and Weiss, M.: Visualizing Early Aspects with Use Case Maps. *To appear in Transactions on Aspect-Oriented Software Development*.
13. Mussbacher, G., Amyot, D., Araújo, J., Moreira, A., and Weiss, M.: Visualizing Aspect-Oriented Goal Models with AoGRL. *2nd International Workshop on Requirements Engineering Visualization (REV'07)*, New Delhi, India (October 2007)
14. Mussbacher, G., Amyot, D., Whittle, J., and Weiss, M.: Flexible and Expressive Composition Rules with Aspect-oriented Use Case Maps (AoUCM). *10th International Wksh. on Early Aspects (EA 2007)*, Vancouver, Canada (March 13, 2007)
15. Roy, J-F.: *Requirements Engineering with URN: Integrating Goals and Scenarios*. MSc. thesis, OCICS, University of Ottawa, Canada (2007), http://www.softwareengineering.ca/jucmnav (accessed November 2007)
16. *URN Virtual Library*. http://www.usecasemaps.org/pub (accessed Nov. 2007)
17. *User Requirements Notation (URN) – Language Requirements and Framework*, ITU-T Recommendation Z.150. Geneva, Switzerland (February 2003), http://www.itu.int/ITU-T/publications/recs.html (accessed November 2007), http://www.UseCaseMaps.org/urn (accessed November 2007)
18. Whittle, J., Moreira, A., Araújo, J., Jayaraman, P., Elkhodary, A., and Rabbi, R.: An Expressive Aspect Composition Language for UML State Diagrams. *Model Driven Engineering Languages and Systems, 10th International Conference, MODELS 2007*, LNCS 4735, pp. 514-528, Springer (2007)
19. Yu, E.: *Modeling Strategic Relationships for Process Reengineering*. Ph.D. thesis, Department of Computer Science, University of Toronto, Canada (1995)
20. Yu, Y., Leite, J. C. S. d. P., and Mylopoulos, J.: From Goals to Aspects: Discovering Aspects from Requirements Goal Models. *12th International Requirements Engineering Conference (RE'04)*, Kyoto, Japan (September 2004)